KitAss - Kitchen Assistant Robot

Oguzhan Cesur
LMU Munich
Munich, Germany
o.cesur@campus.lmu.de

Sarp Cagin Erdogan
LMU Munich
Munich, Germany
sar.erdogan@campus.lmu.de

Ozan Tanriverdi
LMU Munich
Munich, Germany
tanriverdi.ozan@campus.lmu.de

ABSTRACT

Having a helper in the kitchen is a crucial need for many groups of people, either from old age or because of a possible impairment. With our project, we present a prototype that can assist the lives of these groups in the kitchen. Kitchen Assistant (KitAss) is an everyday assistant that is designed to assist its users by automating the seasoning process. The assistant takes the user's voice command as input, where the user specifies the type and amount of the desired seasoning. The kitchen assistant then detects the location of the spice in its working area and picks it up. The picked-up spice is brought up to the target location underneath, where a weight scale sensor has been placed. Using a PID controller, the robot starts to toss the spice and, at the same time, calculates the angle of the joint responsible for the tossing by utilizing the real-time feedback received from the weight scale sensor. When the desired amount has been reached, tossing stops, and the spice container is brought to its original location. In the following section, the motivation behind the project, how the assistant works in each step, and how the project can be extended further will be explained in more detail.

CCS CONCEPTS

• **Computer systems organization → Robotic control**.

KEYWORDS

Human-Robot-Interaction, Speech-to-Text, AprilTag, PID-Controller, ROS, MoveIt,

ACM Reference Format:
Oguzhan Cesur, Sarp Cagin Erdogan, and Ozan Tanriverdi. 2018. KitAss - Kitchen Assistant Robot. In . ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The world's population is getting older, which emphasizes how urgently we need technological developments that help older people maintain their independence, especially in the kitchen. KitAss represents a significant effort in this direction, utilizing robotics to simplify kitchen tasks and improve people's health management. Our project focuses on addressing the challenges rooted mainly in impaired vision of older adults and managing dietary health, particularly sodium intake. We try to achieve this by leveraging techniques from different disciplines and utilizing different sensory and input information.

The main problem we want to solve is the normalization of the sodium intake in risk groups. High sodium consumption is a known risk factor for several conditions prevalent in older populations, such as hypertension, heart disease, and kidney dysfunction. By automating the process of seasoning, we aim to find a solution that supports the health and well-being of our users. Enhancing human-robot interaction is central to this project, aiming to create an intuitive and user-friendly system that older adults can use easily, thus bridging the gap between advanced technology and everyday practicality in independent living.

We developed KitAss, combining the utility of voice commands for precise measurement, and automated the handling of the seasoning process. The project integrates the following key features:

- **Voice Assistant**: The voice assistant enables users to interact with the robot. This interaction method is crucial, especially for older adults with vision impairment. Users can specify the type and amount of spice they need through simple voice commands, making cooking more accessible and enjoyable.
- **Object Detection**: Our robot utilizes object detection using AprilTags [6] to facilitate the accurate locating and handling of spices. This capability allows the robot to identify and locate spice containers in the kitchen without the need to have spice containers at predefined locations. It's particularly beneficial for older adults, enabling them to avoid the physical strain of searching for and retrieving spices.
- **Scale**: Precision in spice measurement is a big deal, especially when managing sodium intake. We integrated a weight scale sensor into our pipeline, providing real-time feedback to ensure that the robot dispenses the correct amount of spice every time.
- **Dynamical Movement**: The robot's design includes dynamic movement capabilities, enabling it to navigate the kitchen environment pick up and return spice containers to their original locations. We also considered possible real-world problems and developed ways to deal with them, i.e., by restricting the robot movements, which wouldn't hold spice containers horizontally. This functionality reduces the need for physical exertion from the user, making the cooking process safer and more enjoyable.

2 NODE SYSTEM

The robot uses a ROS node system in the background for external I/O and internal communication. The necessary inputs, namely voice, camera, and scale values, are regularly provided to the core node. This core handles these values and publishes the corresponding commands to be taken into action: TTS feedback and robot

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

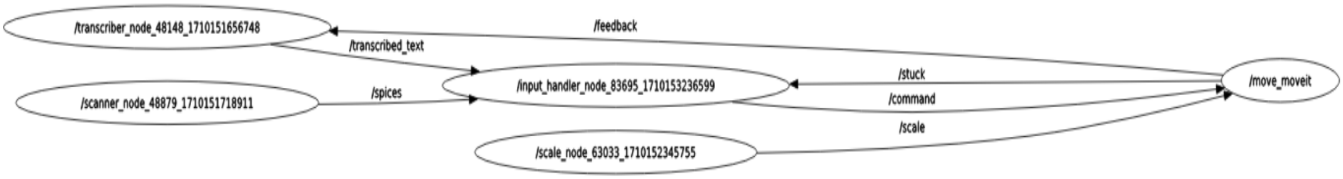

Figure 1: Node Graph

movements. An indexing system is utilized for the published messages to ensure that every unique command is handled correctly, without duplicate or missing actions. 1

3 VOICE INPUT

The transcriber node is a threshold-based audio recording and transcription module within the system's architecture. It continuously monitors the designated input channel of the device, awaiting audio levels that surpass a predefined threshold. This threshold serves as an indicator of potential user voice input. Upon activation, the node commences recording the audio stream. The module implements a cessation criterion to promote efficient resource utilization and focus on relevant data. If the audio level remains below the threshold for a period of two seconds, the recording process terminates. The captured audio is then stored locally in preparation for subsequent processing. The system employs the "insanely-fast-whisper" [2] model to facilitate the transcription process. This model demonstrates proficiency in transcribing human speech from audio recordings, yielding a textual representation of the user's input. The resulting text is indexed to enhance searchability and forwarded to the input handler component. The input handler analyzes and interprets transcribed commands, enabling the system to execute appropriate actions.

4 CAMERA INPUT

Our system utilizes AprilTags for continuous and dynamic spice tracking. The scanner node uses a video capture device to tirelessly monitor the environment. Each frame of the resulting video stream is analyzed for the presence of AprilTags. These tags are linked to a local JSON spice-tag database that identifies the corresponding spice for each tag and designates a reference tag. The AprilTag system excels at providing 3D transformation data (position and orientation) from the 2D tag images captured by our camera. Using the reference tag, we calculate the relative distance between it and each spice container. Adjustable robot-reference-difference values are incorporated to account for the distance between the robot and the reference tag. This results in a dynamic position-gathering system that continuously outputs the positions of spice containers relative to the robot's body, as long as the distance between the robot and the reference AprilTag is known and provided.

5 INPUT HANDLER

The input handler receives the latest position of the spice containers and the voice input content given by the user. It then checks whether the keyword 'assistant' is in the input to determine if the speech was an attempt to communicate with the robot. If that's the case, it uses an algorithm to extract the information and command given by the input based on the two main factors: spice and amount. Suppose a single amount of a single spice type exists in the provided rack. In that case, it converts the input into unified commands for the robot using the information extracted from the command and its local database (spice type - Apriltag relations and conversions for different measurement units like teaspoons, etc.). Every unique command is indexed and published to the robot movement node, including action and the value, such as the position of the spice container and the amount requested in grams. If the user provides an invalid input, our TTS module calculates and gives corresponding feedback to inform the user about the situation. This module also depends on the feedback from the movement node to see if the robot is currently busy on a task or not to toggle its state and accept further requests from the user.

6 SCALE

As previously introduced, we integrated a weight sensor to regulate the spice amount the robot hand was tossing. If the measured spice weight comes close to or exceeds the desired amount, the robot hand stops the tilting motion and brings the spice bottle back to a horizontal state.

For this purpose, the sensor of choice was a Mini Scales Unit by M5Stack [8]. Due to the need for more documentation by both the robot and the sensor manufacturers, making scale readings was a huge problem. The first intuition was connecting the sensor to the 4-pin connector directly on the robot with the hopes that the robot would provide some means to make these readings possible since both the robot and the sensor were M5Stack products. Unfortunately, this wasn't the case, and the solution we used was making scale readings via an external Arduino Uno [1] board. Luckily, M5Stack provided some code to make readings usable after some modifications.

To set this up, we placed a weight sensor under the target location where the robot was supposed to toss the spices. Our vision was that the robot should be able to determine the amount of the tilting motion robustly, independent of how filled the spice bottle is or how high/low the desired spice amount is. This motion meant that we needed constant feedback from the sensor, which the robot controller could then use to determine the angle of the joint, which we determined to be responsible for the tilting motion.

To realize this, we implemented a PID Controller [7], which took values such as the desired value and the current scale reading and output the angle value for the specified joint. We transferred

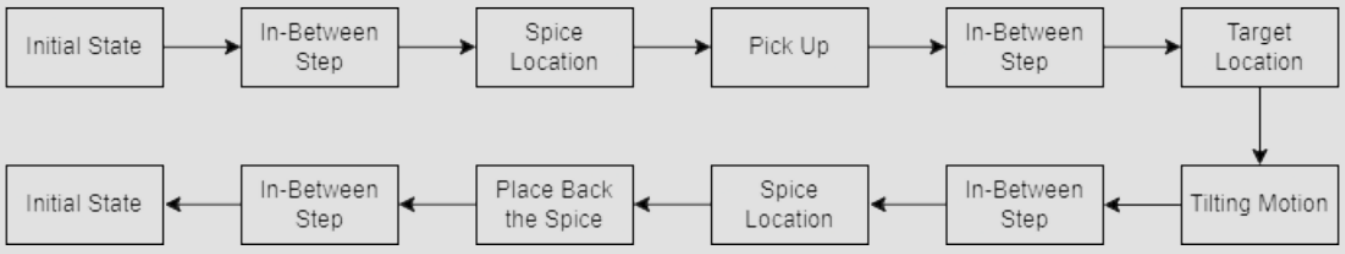

Figure 2: The general movement pattern

scale values to the PID Controller with a high frequency, so the angle values were determined at the same frequency, resulting in an almost continuous joint movement during the tossing process.

After some quick tests, we saw that the scale values were almost always non-zero, even if the robot tossed no spice on them. Because the node architecture of ROS would result in the PID Controller constantly regulating the angle value and the robot arm trying to toss if it's not supposed to, to prevent this, we added a trigger value that would only allow the robot to toss after the robot positioned gripper is above the target location.

The PID Controller offers some hyperparameters that can be tuned for an even more precise and robust joint movement. Performing such a hyperparameter tuning would be highly time-consuming since it would depend on actual robot movements and our own observation of the robot's behavior. So, this hasn't been performed because we found behavior achieved with the values at hand to be within acceptable range.

7 MOVEMENT

We retrieved information from the scale and used the object detection components to perform the robot's actions. The previous chapter explained how we converted the scale values to actions via the PID Controller. We transfer the coordinates of the desired spice from the object detection component to the movement component. The camera and the robot possess different coordinate systems. So, a transition is required, performed by the scanner node beforehand. Thus, the controller node can directly work with the goal coordinate it receives.

In the initial development phase, we used the PyMyCobot library [5] for the robot movement. However, it became clear later that PyMyCobot wouldn't be sufficient and robust enough for the increasing complexity of the project since dynamic spice locations via object detection were planned and later implemented. So, the solution of choice was the MoveIt Motion Planning Framework [3]. Two desirable features of MoveIt were its built-in motion planner and movement constraints [4], which we could define on our own.

Because of spices' high granularity and fluid-like nature, the gripper needed to be kept horizontal during every movement where the gripper was holding a spice bottle. This movement meant that the shortest or the most efficient path would not always be an applicable movement. We implemented a constrained movement function using the Orientation Constraints to realize this. We empirically determined the rotations we wanted to allow or restrict and created our own Orientation Constraint with them. This constraint restricts the path planner from finding a solution that satisfies the defined constraint using a constraint solver. Finally, it wasn't necessary to impose this constraint for the movements without the spice bottle, so we implemented another movement function without any constraints.

Similar to the scale component, we suffered and lost time due to the lack of documentation for MoveIt. The constraints were especially under-documented, which prevented us from implementing any other constraints to mitigate some other undesired movements of the robot, even though we had the ideas developed for them. Another constraint issue was the tolerance parameters we could have used to find a compromise between robustness and a high path-finding success rate. However, these were also under-documented again; testing them empirically would be too time-consuming.

During our tests, we realized that the path planner sometimes was unable to find a solution (viable path), i.e., between the spice and target locations. To reduce the complexity of this problem and help our path planner, we wanted to define an "In-Between Step" for the movements. That was an easy-to-reach location, which we determined based on our own observations. We used this intermediate step between any bidirectional movement from the spice to the target location. The general movement pattern of the robot looks like the following: 2

8 LIMITATIONS AND FUTURE WORK

The journey of developing KitAss showed us the importance of user-centric design, especially when creating technology for populations with specific health and ergonomic needs. Problems encountered in making sensor readings and optimizing the robot's movements to safely and robustly interact in a kitchen environment provided us valuable insights into the challenges faced when designing assistive technologies. In future work, we plan to increase user accessibility even more by extending the project with features where the user only specifies what dish is prepared, and KitAss then, using its preset spice-dish combinations, handles the seasoning on its own. Another important future development we are planning is integrating KitAss into other Smart Home applications where the consumption data of an individual can be utilized to draw valuable health-related conclusions. This future work will develop intuitive interfaces that can be easily integrated into people's routines, ensuring that the technology works seamlessly, where the users don't have to change the way they use KitAss.

Ultimately, this project not only represents a significant step forward in using robotics to improve the quality of life for older adults

but also shows the potential of combining different technologies for a higher-level goal, such as health management.

REFERENCES

[1] [n. d.] Arduino uno. (). https://store.arduino.cc/products/arduino-uno-rev3.

[2] [n. d.] Insanely fast whisper. (). https://github.com/Vaibhavs10/insanely-fast-whisper.

[3] [n. d.] Moveit documentation. (). https://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/move_group_python_interface/move_group_python_interface_tutorial.html.

[4] [n. d.] Moveit move constraints. (). https://answers.ros.org/question/304770/moveit-joint-space-constraints-in-python/.

[5] [n. d.] Pymycobot documentation. (). https://github.com/elephantrobotics/pymycobot.

[6] Andrew Richardson, Johannes Strom, and Edwin Olson. 2013. AprilCal: assisted and repeatable camera calibration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. (Nov. 2013).

[7] [n. d.] The pid controller & theory explained. (). https://www.ni.com/en/shop/labview/pid-theory-explained.html.

[8] [n. d.] Weight scale sensor. (). https://docs.m5stack.com/en/unit/Unit-Mini%20Scales.